Terence Henriod

HW 13

CPE 301: Microprocessors System Design

December 5, 2013

**Perform a WEB search for VERILOG and VHDL programming tutorials and pick two for each language which you think are useful. Also, briefly examine the history of both languages and examine their similarities and differences.**

1. **Print out 1 representative page from each of the best 4 tutorials (2-Verilog and 2-VHDL) including URLs.**
2. **Reference Designer**

Main URL: <http://www.referencedesigner.com/tutorials/verilog/verilog_01.php>

Sample Content URL: <http://www.referencedesigner.com/tutorials/verilog/verilog_02.php>

Let us start with the design of a simple comparator. This will also be our "Hello World" of the Verilog. Let us take a look at the following table which describes the behavior (We are using American spelling behavior in place of behaviour - sorry about that) of a comparator circuit.

**Table: A one bit comparator**

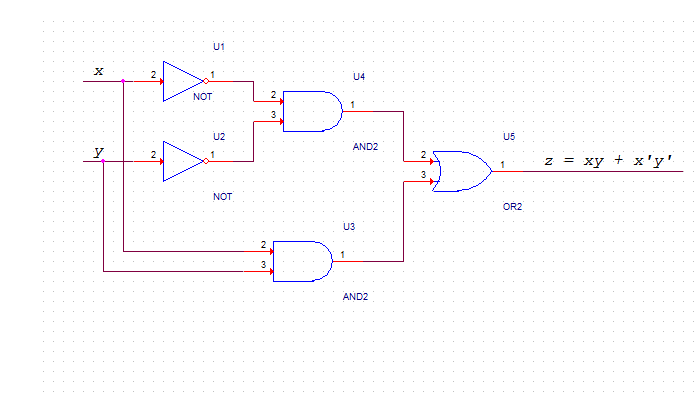
|  |  |  |
| --- | --- | --- |
| **Input x** | **Input y** | **Output z** |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Basically when both the inputs x and y are same, the output z is 1. When the inputs are unequal, the output is 0.   
We can describe the circuit using **AND**, **NOT** and **OR** gates using the following equation.

assign z = (~x & ~y) |(x & y); 

where ~x and ~y represent the complements of x and y respectively. 

The following shows a circuit that implements this logic.

   
  
And here is the verilog code that implements this logic 

|  |
| --- |
| 1. **module** comparator( 2. **input** x, 3. **input** y, 4. **output** z 5. ); 7. **assign** z = (~x & ~y) |(x & y); 9. **Endmodule** |

We will try to make you understand what Verilog is - in a matter of one day. At least you should be able to compile and run verilog code (Kind of Hello World of verilog). We hope that this is something you will be able to achieve within next few pages. Throughout this tutorial, we will present you enough examples and exercises so that you have a good grip over the language as well as the verilog concepts.   
  
While Verilog has concurrent blocks executing in parallel, it is still similar to software programming language like C.   
  
If you have closely watched the schematics above and the verilog code below it, you must have appreciated how verilog simplifies the process. Before the advent of Verilog, everything was done using schematics. The schematics were error-prone, difficult to verify, and had long process of design, verfification, fix, redesign and re verify and so on.   
  
With Verilog, the whole dimension and process of hardware circuit design changed. This provided a new way of looking at designing the circuiy. Verilog design is more like a software programming, but, you must also have a strong understanding of the circuit that works behind the code.   
  
Let us now understand the code. Take a look at 

|  |
| --- |
| 1. **module** comparator( 2. **input** x, 3. **input** y, 4. **output** z 5. ); |

Verilog consists of modules. Inside the modules, we have a list of ports ( or pins). A port can be an input port or an output port depending upon its direction. A pin can also be defined as bidirectional using inout.   
  
The direction can also be specified out of the module. This code is equivalent to the previous code. In Verilog 1995, this was the only option. The Verilog 2001 allows port direction to be defined with the module declaration itself. 

|  |
| --- |
| 1. **module** comparator( 2. x, 3. y, 4. z 5. ); 7. **input** x; 8. **input** y; 9. **output** z; |

Let us now take a look at the assign statement 

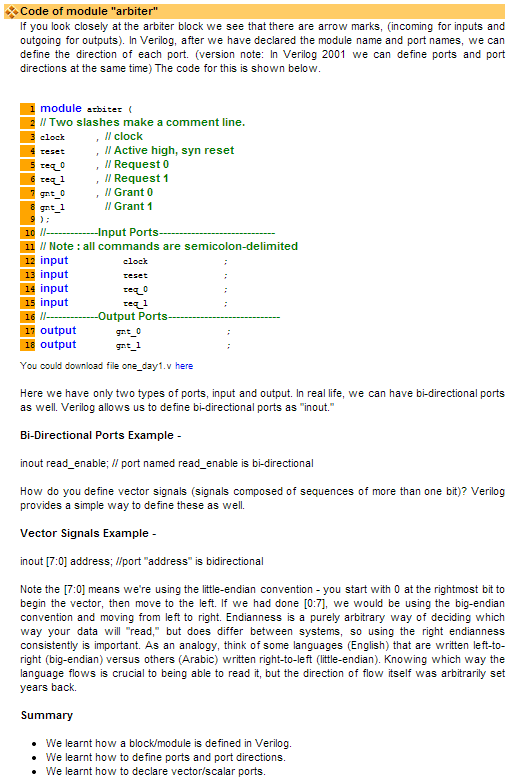
assign z = (~x & ~y) |(x & y);

This assign statement implements a combinational logic. An assign statement is used for modeling only combinational logic. The statement in the assign statement is executed continuously ( as against those that trigger on a clock). An assign statement is also called 'continuous assignment statement'.   
  
This statement implements the comparator logic that we had shown earlier in the schematics.

1. **Asic-World**

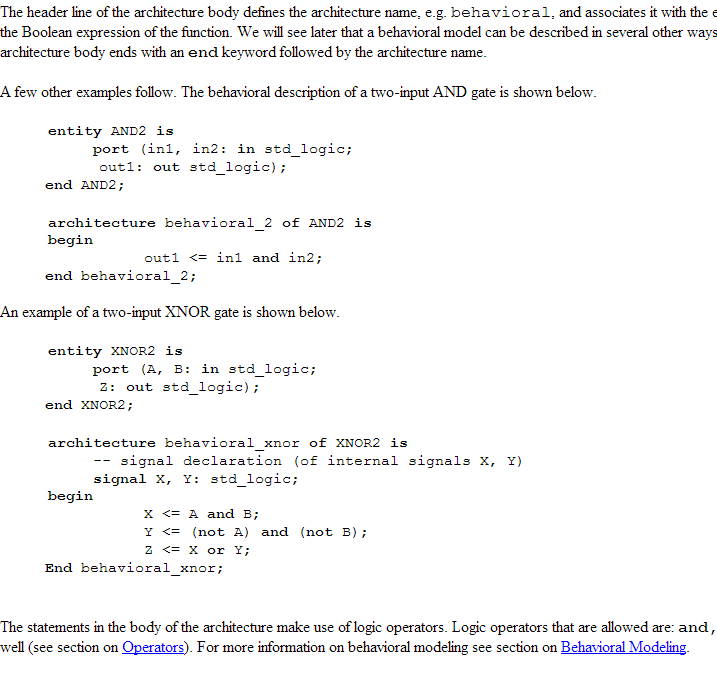
Main URL: <http://www.asic-world.com/verilog/veritut.html>

Sample Content URL: [http://www.asic-world.com/verilog/verilog\_one\_day1.html#Code\_of\_module\_"arbiter"](http://www.asic-world.com/verilog/verilog_one_day1.html#Code_of_module_)



1. **University of Pennsylvania**

Main URL: <http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html#_Toc526061340>



1. **Asic-World**

Main URL: <http://www.asic-world.com/vhdl/tutorial.html>

Sample Content URL: <http://www.asic-world.com/vhdl/first1.html#Hello_World_Program>



1. **Write a 2 page report, in your own words, describing what you learned and why you think your chosen tutorials are good. Include references (URLs) to any really good web pages describing or comparing the languages. Do NOT hand in more than 8 pages**.

I learned a lot from this assignment, which may or may not say a lot because this assignment was about finding tutorials to learn HD Languages. There was an entire new world to learn. What was not surprising was that the tutorials preached the need for “top-down” design. Even in hardware where we deal with the smallest units of computing, we still should have a “big picture design” before we worry about the small stuff. [1]

I thought it was interesting to see that data types need to be defined in terms of how many bits they require, and that Verilog supports both big-endian and little-endian bit arrays. The expectation with languages is that things like data types are set in stone, but I suppose if you’re going to deal with things at the hardware level, you should be able to really control things at the hardware level. [1]

I learned that Verilog is a dataflow language, meaning that actions happen more or less concurrently. Instead of using a more sequential programming paradigm, Verilog uses modules that interact with one another that begin operation once all of their inputs become valid. [2]

I learned that Verilog supports “blocking” and “non-blocking” assignment. Blocking assignment is the typical kind of assignment that is normally used in most programming languages, the ‘=’ operator. This operator immediately performs the stated assignment in the order of the given sequence. [2]

Based on the code snippets I saw between the Verilog and VHDL languages, I think I’ve learned that based on first impressions I will prefer coding in Verilog. The ASIC-World author seems to agree, as in one of the pages of the Verilog tutorial he praises Verilog’s small number of reserved keywords/language parts. Also, from the code snippets I’ve seen, I think that the language of Verilog feels more intuitive and easy to handle, with fewer keywords with new definitions (e.g. the line with keyword **assert** in it, I’m sure this assert is different from the one in C/C++). [1]

I think that the ASIC-World tutorial will be the best of the tutorials that I found. It contains the most graphics, code and otherwise, which will be helpful in following the author’s logic, even if his English is not the best. There are truth tables and logic circuit diagrams that go with any code that will be written, and each segment of code has its own graphic, displaying the code as it might appear in a compiler, making it very clear what kind of code is being used for which operations. Also, the instructor said it was a really good website in class.

However, this is not to say that the other tutorials I found will be of poor quality. I made sure to choose tutorials with graphics and code snippets. I think the “Reference Designer” tutorial will be a good competitor for the ASIC-World ones.

Surprisingly, I think the Penn State tutorial that I found will be the least helpful. I find this the least helpful because it seems to be the only one written by a native English speaker/writer (although the English grammar in the other tutorials really isn’t horrible, it’s actually very understandable, save a few grammar and spelling errors). It seems to lack some of the graphics and supporting text to accompany code snippets that the other tutorials seem to have an abundance of.

In general though, I think that this collection of tutorials will be a good start to my journey in learning Hardware Description languages and creating Application Specific Integrated Circuits (or ASICs).

# Works Cited

|  |  |
| --- | --- |
| [1] | D. K. Tala, "Verilog Tutorial," ASIC-World, 1 August 2013. [Online]. Available: http://www.asic-world.com/verilog/veritut.html. [Accessed 4 December 3013]. |
| [2] | Miscellaneous, "Verilog - Wikipedia, the free encyclopedia," 19 November 2013. [Online]. Available: http://en.wikipedia.org/wiki/Verilog. [Accessed 4 December 2013]. |